JUNE 74

Popular Computing

BOX 272 CALABASAS, CA 91302

15

VOL 2 NO 6

A	8	9	7	6	6	3	4	8	5	2	1	
	9	8	5	5	3	6	7	1	5	9	6	
	2	6	1	1	7	3	2	4	2	3	6	
	4	5	8	6	7	2	1	5	7	4	4	
	1	9	9	3	8	2	3	4	9	7	8	
	3	7	9	5	8	0	1	7	8	2	5	
	4	8	1	7	9	5	5	9	4	3	3	
	2	8	6	2	2	6	5	3	7	2	4	
	3	4	8	4	9	1	7	8	4	1	5	
	1	9	7	7	3	6	3	1	2	5	9	
	6	3	6	8	4	5	6	6	7	9	8	B

Pocket Calculator Game

Game

The POCKET CALCULATOR GAME is for two players, A and B. Players enter the board at the places marked, and proceed in turn, with the following move rule:

Advance 1, 2, 3, 4, or 5 squares on the zigzag pattern. The score for each move is given by:

$$N + \sqrt{N}\sqrt{C} - C_0$$

where N is the number of squares advanced; C is the contents of the square landed on; and C_0 is the contents of the square the opponent last landed on. For the two initial moves, C_0 is zero.

The game ends when one player overtakes or passes the other on the board. Ten points per square is added to the score of the player who passes the center square, outlined in red.

EACH PLAYER KEEPS SCORE FOR HIS OPPONENT. Two markers, such as those included with the initial press run of this issue, facilitate keeping track of each player's position on the board. The accompanying table of products of square roots is an aid to keeping score.

The loser of one game starts first in the subsequent game. The sides of the board are alternated in successive games.

POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in the United States is $18 per year, or $15 if remittance accompanies the order. For Canada and Mexico, add $4 to the above rates. For all other countries, add $6 to the above rates. Back issues $2 each. Subscriptions may begin with any issue. Copyright 1974 by POPULAR COMPUTING.

Publisher: Fred Gruenberger
Editor: Audrey Gruenberger
Associate editor: David Babcock

Contributing editors: Richard Andree
Paul Armer
Daniel D. McCracken
William C. McGee

Advertising manager: Ken W. Sims
Art director: John G. Scott

	1	2	3	4	5	6	7	8	9
1	1.0000	1.4142	1.7321	2.0000	2.2361	2.4495	2.6458	2.8284	3.0000
2	1.4142	2.0000	2.4495	2.8284	3.1623	3.4641	3.7417	4.0000	4.2426
3	1.7321	2.4495	3.0000	3.4641	3.8730	4.2426	4.5826	4.8990	5.1962
4	2.0000	2.8284	3.4641	4.0000	4.4721	4.8990	5.2915	5.6569	6.0000
5	2.2361	3.1623	3.8730	4.4721	5.0000	5.4772	5.9161	6.3246	6.7082

N-Series 4 & 8

Log 4	.60205999132796239042747778944898605353637976292422
Ln 4	1.3862943611198906188344642429163531361510002687205
$\sqrt[3]{4}$	1.5874010519681994747517056392723082603914933278998
$\sqrt[5]{4}$	1.3195079107728942593740019712296401330334690131934
$\sqrt[7]{4}$	1.2190136542044754409116910025925608572774119358599
$\sqrt[10]{4}$	1.1486983549970350067986269467779275894438508890978
$\sqrt[100]{4}$	1.0139594797900291386901659996282304258363540227495
$\tan^{-1} 4$	1.3258176636680324650592392104284756311844406013064
Log 8	.90308998699194358564121668417347908030456964438633
Ln 8	2.0794415416798359282516963643745297042265004030808
$\sqrt[5]{8}$	1.5157165665103980823472598013064452386812835429781
$\sqrt[7]{8}$	1.3459001926323561319428326037509419043662470267775
$\sqrt[10]{8}$	1.2311444133449162844993930691677431098761377611008
$\sqrt[100]{8}$	1.0210121257071932497640951747830643735410879532737
$\tan^{-1} 8$	1.4464413322481351841999668424758804165254145079177

In the interest of completeness, the entries in the N-Series for 4 and 8 are given here.

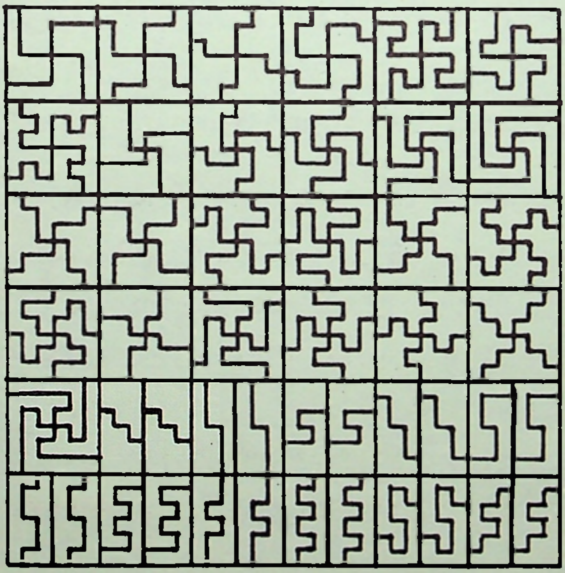
Parkin

Problem 15 called for the number of ways in which an 8 x 8 checkerboard could be cut into four congruent pieces, cutting along the lines of the checkerboard. The cover of issue No. 7 showed the 37 unique ways this could be done for a 6 x 6 board.

Thomas R. Parkin, Vice President, Control Data Corporation, using an algorithm given below implemented on a CDC 6600, has extended the known results as follows:

board size	number of ways
2	1
4	5
6	37
8	782
10	44240

with an estimate for the 12 x 12 board of 7750000. Mr. Parkin estimates that the 12 x 12 case, using his algorithm on the CDC STAR, would take 5 to 10 minutes, and the 14 x 14 case from 30 to 60 hours. The following is reproduced from his letter.

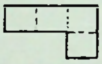


It might be well, first, to show some comparison numbers to support the seemingly large values for C{N}. We observe that each shape which cuts the square checkerboard into four congruent pieces is a polyomino of order N^2. Therefore, we might suspect that the number C{N} would be less than the corresponding number of polyominoes of the corresponding number of squares, since, of course, there are poloyominoes of any given size {≥4} which cannot be used to divide the square array into four congruent pieces, i.e., the 1 x 16 shape. Let's look at a comparative table:

n	1	2	3	4	5	6
N=2n	2	4	6	8	10	12
N^2	4	16	36	64	100	144
$N^2/4=M$	1	4	9	16	25	36
P{M}	1	5	1285	13079255	$\approx 2.03\times10^{12}$	$\approx 5\times10^{18}$
C{N}	1	5	37	782	44240	$\approx 7.75\times10^{6}$

Thus we see that, in general, C{N} $\ll$ P{N^2}.

On the other hand, one might argue that even though all the P{N^2} cannot be used, one or more might be used several times;

for example, the  piece for N=4 can be used two different

ways to yield two countably distinct divisions of the board into four congruent pieces. {Incidentally, if I were setting the rules, I would exclude this case since the stated problem is "to cut the square checkerboard into four congruent pieces", and once these pieces have been cut, I maintain their origin is no longer germane; however, this is not a problem except when N ≡ 0 mod 4 and it can be argued that the emphasis is on 'cut', and not the shape of the cut piece, so I won't press the point.} Even so, an examination of the counts of P{N^2} by symmetry type also shows that pieces possessing both 90^o rotational symmetry at one point and 180^o symmetry at another are vanishingly few, thus the conclusion, C{N} $\ll$ P{N^2} seems valid.

There are a few observations about the problem which are necessary before one can understand the algorithm which I used for counting the possible cases. First of all, it is necessary to adopt a systematic procedure for enumerating the shapes, and, as a precursor to development of that systematic procedure, it is necessary to adopt some taxonometric scheme for classifying the shapes. Note that there are two classes of cuts which will divide the checkerboard into congruent pieces, and these may be classified as those which cut the square array into four pieces such that the edges of the pieces are 90° rotationally symmetric within the square, and the remaining cases where the square is divided into two rectangles by a line through the center, bisecting two sides, and then the rectangle is divided into two congruent pieces by a line which is 180° rotationally symmetric. Next, note that the pieces may be displayed either obverse or reverse, i.e., mirror images, or they may all be oriented in some canonical fashion.

In my experience with writing programs to deal with two dimensional shapes, I have found that coding the shapes is quite tricky and difficult to work with; however, dealing with the edges of shapes is sometimes simpler. Thus, in this problem I chose to consider the square array of 2N x 2N squares in the plane to be a {2N+1} x {2N+1} square array of lattice points in the plane and trace paths corresponding to the edges of the shapes of interest.

The interior edges of the various pieces then provide for a taxonomy and allow a systematic procedure for generating shapes. Note that every piece in the square case involves an edge which traces a path from the outer perimeter of the square lattice to the center point. Furthermore, this path originates, say, along the upper left border of the square down to the center line, and from no other place. {This is the canonical orientation.} It is these unique paths which connect the border with the center which we enumerate {in the square case} and similarly, with suitable restrictions, for the rectangular case.

We note, of course, that the origin of the paths in the square case ranges over only one-eighth of the border {because of rotation and reflection symmetry}, and over only one quarter of the border in the rectangular case.

Now, of course, the algorithm is simple. Start at each appropriate border point and trace all possible unique paths to the appropriate center. I realize that this statement is descriptive but hardly constructive, so I will further detail the process.

1. Start at a corner, proceed down one side {or across the top}, and select the next point which has not been used as a starting point. Stop after using the center line point {or less than or equal to one-quarter of the top}. After selecting the border point, mark all other border points as "disallowed". Enter the border point in a list at Level 1.

2. Select the next lattice point inside the array, i.e., toward the center, and enter this point at Level 2. Note that there is only one possible choice for the second point on a path, given the first point on the border.

3. Mark the four {or two} symmetrical points to the last selected point as "disallowed". {The purpose of this disallowance is to preclude any point from appearing on more than one path, an obvious impossibility, thus, as each point on a path is selected, we simply check off the corresponding symmetrical points which are then no longer available for extensions of the current path.}

4. Add to the list, at the current level, the coordinates of the 0, 1, 2, or 3 possible points which could possibly be the extension of this path and mark them "unselected". Note that of the four points surrounding a given point, one of them is where the path came from, so that there are, at most, three possible extensions. Furthermore, it is possible to trace a path into a cul-de-sac such that no further extension of that path is possible and the path has not arrived at the center, hence the 0 possibility.

5. If there are further eligible selections at this level, select the next unselected eligible extension point on this path at this level, mark it "selected", and enter it at the next level. If there are no further selections at this level, remove the "disallowed" exclusions for the selected point, and the selected point, at this level. Back up one level. If Level 1 has been reached, all done this border point; if not, repeat this Step 5.

6. Match the selected point against the end point {or points} to see if this path is finished. If not, recursively repeat Steps 3, 4, 5, and 6 until the path terminates or an exit at Step 5 occurs.

7. Tally the terminated path {and note its coordinates, if you have time!}

8. Backtrack one level and repeat at Step 5.

Using this algorithm, suitably modified in implementation to cover both the square and the rectangular cases, and again, suitably modified for the evenness or oddness of N, yields the table of results, Table 1 attached.

➔

N	2N	I,J {Square}	#	I,J {Rect}	#	C{2N}
2	4	1,0	2	1,0	1	5
		2,0	1	0,1	1	
3	6	1,0	12	1,0	4	37
		2,0	9	2,0	3	
		3,0	5	0,1	4	
4	8	1,0	212	1,0	43	782
		2,0	167	2,0	33	
		3,0	146	3,0	24	
		4,0	87	4,0	8	
				0,1	43	
				0,2	19	
5	10	1,0	11030	1,0	1026	44240
		2,0	8774	2,0	821	
		3,0	7538	3,0	670	
		4,0	7229	4,0	550	
		5,0	4522	5,0	243	
				0,1	1026	
				0,2	811	
6	12	1,0	$<1.8 \times 10^6>$	⋮		$<7.75 \times 10^6>$
		⋮		0,1	$<1.25 \times 10^5>$	
		6,0	$<6 \times 10^5>$	⋮		
				0,3	$<5 \times 10^4>$	

<estimated>

The Way to Learn Computing is to Compute

More Parkin

Problem 30, The Web of Fibonacci (PC-10) called for the construction of a succession of triangles whose sides were the square roots of triplets from the Fibonacci sequence (1, 1, 2, 3, 5, 8, 13, 21, 34,...). The first point of the Web lies at $(\sqrt{2},0)$ and the second point at $(\sqrt{2},1)$; the problem was to find the coordinates of the 150th point. A solution by Thomas R. Parkin, Control Data Corporation, points out that one needs the value of F_{151}, where

$$F_0 = 1, \quad F_1 = 1, \quad F_2 = 2, \quad F_3 = 3, \text{ and so on.}$$

F_{151} is a 32-digit number:

$$26099748102093884802012313146549.$$

The 150th point lies on a circle of radius $R = \sqrt{F_{151}}$. The pattern is formed of triangles whose central angle of interest is $A = \text{arctangent } (F_i/F_{i+1})$.

"Thus it remains to compute the 148 A's, accumulate them, remove multiples of 2π, compute X and Y from the remaining angle, and there we are.

"Thus $A = \left\{ \sum_{i=1}^{148} \text{arctangent} \left(F_i/F_{i+1} \right) \right\} \bmod (2\pi)$

$$A = 4.9844 \text{ radians}$$

$$R = \sqrt{F_{151}} = \sqrt{26.09974810 \times 10^{30}}$$

$$R = 5.10879125646 \times 10^{15}$$

$$X = R \cos A = 1.37259283882 \times 10^{15}$$

$$Y = R \sin A = -4.92094879071 \times 10^{15}$$

and we see that the point is just inside the fourth quadrant. All calculations were carried out in double precision Fortran on the CDC 6600 and are accurate to at least 12 significant digits for the final values of X and Y since the 96 bits allow for as much as 28 significant figures in all the intermediate calculations, including the trigonometric ones.

"Finally we note that on approximately the scale of the cover of PC-10, the point in question would require a sheet of paper 200,000 <u>million</u> <u>miles</u> square to be plotted!"

Lake/fence

PROBLEM 51

A rectangular field is to be built with 500 feet of fencing next to a straight river. What are the dimensions of the field, if it is to have maximum area?

The area, A, is X(500 - 2X). Setting the derivative, -2X + (500 - 2X), equal to zero, the length X is readily found to be 125 feet. This is not a computer problem, unless one seeks the required value by a bracketing process, to avoid using the calculus.

If the straight river is replaced by a circular lake of radius one mile, the problem becomes more complex. An analytic solution is still possible, but difficult. See Figure D.

$$H^2 = 5280^2 - (250 - X)^2$$

$$H^2 = 27815900 + 500X - X^2$$

$$T = \tan^{-1}\left\{ (250 - X)/H \right\}$$

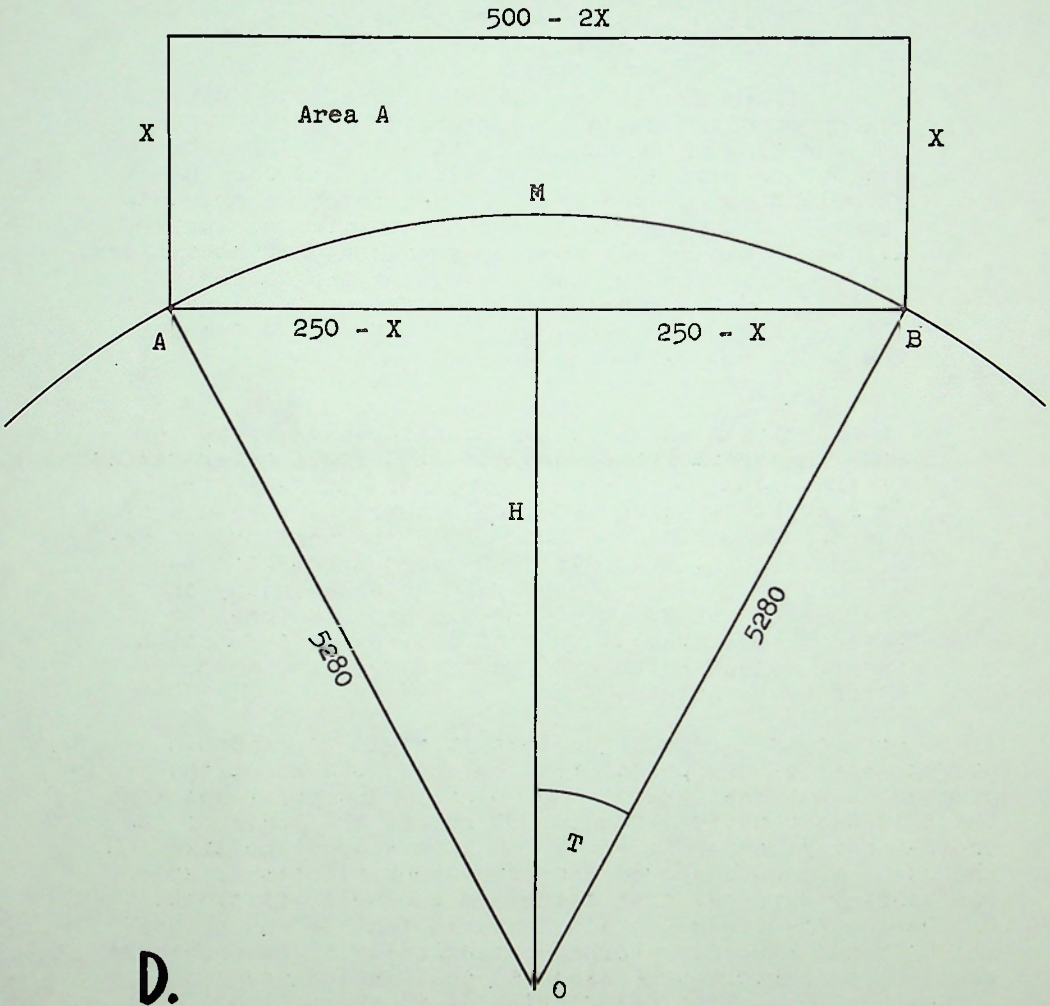
Area A = Rectangle - sector OAMBO + triangle OAB

Area A = $X(500 - 2X) - 5280^2(T) + H(250 - X)$

and X is approximately 126.446...

(A) Flowchart the logic of finding X to many places.

(B) Write a program and run it, to find X to, say, 15 significant digits.



D.

Comment

Irwin Greenwald, Operating Systems Section, Xerox Corporation, writes:

> "In the 'Art of Computing 2' (PC11-4) you state
>
> 'The test procedure should be logically independent of the program itself.'
>
> "I beg to differ. While it is true that the test procedure should be concerned primarily with the function being computed, it will be less than satisfactory if it does not also account for the algorithm being used and idiosyncracies of the code itself. I am primarily concerned with the latter: all too often special cases, anomalous end conditions, etc., are introduced as a result of the manner of coding. Unfortunately, these are frequently known only to the coder, who may not be the person writing the test procedure.
>
> "As a case in point, consider the unnamed mathematician who expended considerable effort to test the precision of the JOSS LOG routine for values of the argument close to zero when, in fact, the difficult precision problems occurred for values of the argument close to one. The 'tester' had no way of knowing that two different algorithms (and two different code sequences) were used depending on whether or not the argument was close to one. Obviously, a crucial part of testing this function was to select a range of test values around the 'crossover' points."

One could hardly argue that it would not improve a program test if the person testing knew more about the program. But that wasn't the point. The point was that the test being devised should not repeat the logic of the program being tested. For a function subroutine like LOG, the test procedure should use some other algorithm, and for testing purposes that algorithm can be inefficient. Of course, the range of values should include end points and critical ranges. Perhaps a logically tight procedure would be to perform some extended calculation based on logarithms (e.g., the calculation of factorial 1000) that can be checked against the known result (PC2-10).

ROBERT TEAGUE

Speaking of Languages

In issue No. 12, page 7, a set of three Fortran programs was presented as a test of how individual compilers interpret the ANSI standard. The programs are given again here:

I.

```
   DIMENSIØN L(3)
   EQUIVALENCE (J,L(2))
   L(2) = 10
   DØ 10 J = 1,3
   K = J
10 CØNTINUE
   WRITE (6,20) L(2)
20 FØRMAT (I7)
   STØP
   END
```

II.

```
   DIMENSIØN L(3)
   EQUIVALENCE (J,L(2))
   L(1) = 46
   L(2) = 47
   L(3) = 48
   DØ 10 J = 1,3
   I = J
   L(I) = J/2 + 4
10 CØNTINUE
   WRITE (6,20) L(2)
20 FØRMAT (I7)
   STØP
   END
```

III.

```
   DIMENSIØN L(3)
   EQUIVALENCE (J,L(2))
   L(1) = 5
   L(2) = 9
   L(3) = 3
   DØ 10 J = 1,3
   L(J) = J*4 + J/2
10 CØNTINUE
   WRITE (6,20) L
20 FØRMAT (3(2X,I7))
   STØP
   END
```

Three questions were asked concerning each of them:

(1) Will it compile?

(2) Will it execute?

(3) What output will be produced?

In the outputs received, all three programs generally compiled and executed. This perhaps is a little surprising, since two of the programs, (2) and (3), reset the index value of the DØ statement. The EQUIVALENCE apparently very successfully hides this fact from the compilers.

The outputs received are summarized in the table on the next page.

Some interesting facts are evident from this exercise. The System 3 compiler apparently loads an index register and uses it as the DØ index without any reference to the variable J in storage. This is evidenced by the fact that although the value of J is changed to 10 during the second loop, and should therefore terminate the DØ since it then exceeds the test value of 3, the loop proceeds to a third iteration giving L(3) a value of 13. All the other compilers stopped after the second iteration, leaving L(3) with its original value of 3. The DØS 'F' level compiler goes a step further, however, and tests for the DØ limit before the loop, since the initial value of L(2) is left unchanged, thus indicating a second iteration of the DØ is never made.

Compiler	Output				
	Program (1)	Program (2)	Program (3)		
1. System 3	3	3	4	3	13
2. B6700, Fortran 2.3	4	6	4	10	3
3. IBM 1130, Eastern Michigan	4	6	4	10	3
4. IBM 360/40, WATFIV	4	6	4	10	3
5. IBM 360/40, 'F' Level DØS	3	5	4	9	3
6. CDC (without optimization)	4	6	Error Message		
7. CDC (with optimization)	1	7	Error Message		
8. CDC Run Fortran	3	5	5	4	3
9. NCR Full Fortran	4	6	5	5	3
10. Univac VMØS Background Fortran	4	6	5	5	3
11. WATFOR Version 1, Level 3	4	6	5	5	3

The two runs on the CDC compiler certainly are of interest since the optimized version produces incorrect values while the unoptimized version produces values you would expect. However, it is notable that this is the only compiler (including WATFOR and WATFIV) that caught the fact that the index was being redefined.

I would be interested in seeing additional outputs to these programs. Of particular interest would be outputs from 360/370 ØS, and Honeywell compilers.

Given below is the DATA DIVISION and PROCEDURE DIVISION for a CØBØL program. Add to the given code the IDENTIFICATIØN and ENVIRØNMENT DIVISIØN entries necessary to run the program on your system. The question to be ascertained is this: what value will be printed for TØTAL?

Readers have continued to work on the Change-Maker problem first presented in PC7-4 with an original solution presented in PC11-8. The first solution contained only 9 Fortran statements. It now appears that the problem can be solved using only 8 Fortran statements and still conform to the ANSI standard requirements. Other short or novel approaches to this problem are welcomed. Send all material to "Speaking of Languages..." at Box 272, Calabasas, California 91302.

```
DATA DIVISIØN.
FILE SECTIØN.
FD  ØUTPUT-FILE LABEL RECØRDS ARE ØMITTED DATA
    RECØRD IS ØUTPUT-LINE.
01  ØUTPUT-LINE.
    02 FILLER PICTURE X(5).
    02 TØTAL PICTURE Z(6).9(5).
WØRKING-STØRAGE SECTIØN.
77  SUM PICTURE 999.
77  CØUNT PICTURE 999.
77  TEST PICTURE 999.
77  I PICTURE 999.
PRØCEDURE DIVISIØN.
P-A.  ØPEN ØUTPUT ØUTPUT-FILE.
      MØVE SPACES TØ ØUTPUT-LINE.
      MØVE ZERØ TØ SUM, CØUNT, TEST.
P-B.  PERFØRM P-C THRU P-D VARYING I FRØM 1 BY 1
      UNTIL I = 8.  GØ TØ P-E.
P-C.  ADD 1 TØ CØUNT.  ADD 1 TØ SUM.
      IF I IS NØT EQUAL TØ 4 ØR 5 GØ TØ P-D.
      ADD I TØ SUM.  ADD 1 TØ TEST.
P-D.  EXIT.
P-E.  CØMPUTE TØTAL = SUM / (CØUNT - TEST).
      WRITE ØUTPUT-LINE.  CLØSE ØUTPUT-FILE.
      STØP RUN.
```

Desk Calculator Reviews

The Unisonic 739SQ (Unisonic, 16 West 25th Street, New York 10010) is a battery-operated, 8-digit pocket calculator with square root, reciprocals, % function, and one word of accumulating storage. On the rating scale published in issue No. 10, using the $68.88 price for which the machine can be obtained, the rating is 8.71.

The Unitrex 800R (Unitrex of America, Inc., 11846 East Washington Boulevard, Whittier, California 90606--sold through Montgomery Ward and Company) is quite similar to the Unisonic described above, but is a desk machine. It has square root, but no extra storage. Besides battery operation, an AC adapter can be obtained. At $59.88, its rating is 7.52.

The ICP 537 ESR (ICP Ltd., New York 10001) is a scientific, battery-operated pocket calculator made in Taiwan. It is an 8-digit machine with floating point (but not scientific notation). There is one word of storage, and the following functions: log, ln, 1/x, e^x, square root, sine, cosine, tangent, arcsine, arccos, arctan, and the power function. If obtainable at $150, its rating is 9.33.

15

Log 15	1.176091259055681242081289008530622282431938982728587323519438179178120963509236613556041103529430 13
Ln 15	2.708050201102210065996004570148713344173091912091267173647342225111673280926266731503749632906911 70
$\sqrt{15}$	3.872983346207416885179265399782399610832921705291590826587573766113483091936979033519287376858673 52
$\sqrt[3]{15}$	2.466212074330470101491611323154589042735484486628053760178787410293376952922897463216298702664346 05
$\sqrt[5]{15}$	1.718771927587478777013521452044409157135458917951753676042921451160068360239063859897620286909505 08
$\sqrt[7]{15}$	1.472356700180346923712790000974024232796166407546775763468282214839388265538123535028083034986501 55
$\sqrt[10]{15}$	1.311019423039749925204556407052804330732016434783535393106126919702334728563440892608633987472376 21
$\sqrt[100]{15}$	1.027450511266726962344780262015611676531612360274458607800034160083740782775935880603768942996443 42
e^{15}	3269017.372472110639301855046091721315505738543820034206629562773242021332748879132969874112 29
π^{15}	28658145.969387998453378821971660543332990696190337284934179898601671305896565088505771141500
$\tan^{-1} 15$	1.504228163019072815032674997345780375000711469862093540786515830933074110674576709932259746885691 16
15^{100}	4065611775352152373972797075670416710103878906323797634290517698787563831961701377171118109321745578199625015258789062 5

N-Series